TECHNICAL REPORT RD-GC-92-30

REAL TIME EXECUTIVE FOR MISSILE SYSTEMS MC68020 ADA INTERFACE

Phillip R. Acuff and
Wanda M. Hughes
Guidance and Control Directorate
Research, Development, and Engineering Center

and

On-line Applications Research Corporation
3315 Memorial Parkway SW
Huntsville, AL 35801

MAY 1992

U.S. ARMY MISSILE COMMAND
Redstone Arsenal, Alabama 35898-5000

Approved for Public Release.

DESTRUCTION NOTICE

FOR CLASSIFIED DOCUMENTS, FOLLOW THE PROCEDURES IN DoD 5200.22-M, INDUSTRIAL SECURITY MANUAL, SECTION II-19 OR DoD 5200.1-R, INFORMATION SECURITY PROGRAM REGULATION, CHAPTER IX. FOR UNCLASSIFIED, LIMITED DOCUMENTS, DESTROY BY ANY METHOD THAT WILL PREVENT DISCLOSURE OF CONTENTS OR RECONSTRUCTION OF THE DOCUMENT.

DISCLAIMER

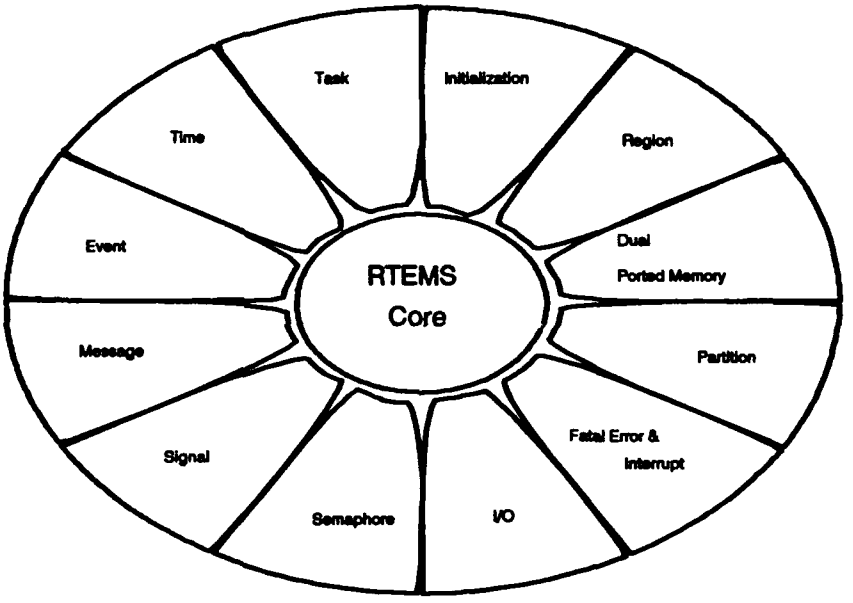
THE FINDINGS IN THIS REPORT ARE NOT TO BE CONSTRUED AS AN OFFICIAL DEPARTMENT OF THE ARMY POSITION UNLESS SO DESIGNATED BY OTHER AUTHORIZED DOCUMENTS.

TRADE NAMES

USE OF TRADE NAMES OR MANUFACTURERS IN THIS REPORT DOES NOT CONSTITUTE AN OFFICIAL ENDORSEMENT OR APPROVAL OF THE USE OF SUCH COMMERCIAL HARDWARE OR SOFTWARE.

Real Time Executive for Missile Systems

MC68020 Ada Interface



U.S. ARMY MISSILE COMMAND
Redstone Arsenal, Alabama 35898-5254

Release 1.31
December 1991

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION / AVAILABILITY OF REPORT Approved for Public Release
2b. DECLASSIFICATION / DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S) Technical Report RD-GC-92-30	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION Guidance & Control Directorate RD&E Center 6b. OFFICE SYMBOL (*If applicable*) AMSMI-RD-GC-S	7a. NAME OF MONITORING ORGANIZATION
6c. ADDRESS (*City, State, and ZIP Code*) Commander, U.S. Army Missile Command ATTN: AMSMI-RD-GC-S Redstone Arsenal, AL 35898-5254	7b. ADDRESS (*City, State, and ZIP Code*)
8a. NAME OF FUNDING / SPONSORING ORGANIZATION 8b. OFFICE SYMBOL (*If applicable*)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c. ADDRESS (*City, State, and ZIP Code*)	10. SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO. / PROJECT NO. / TASK NO. / WORK UNIT ACCESSION NO.
11. TITLE (*Include Security Classification*) Real Time Executive For Missile Systems MC68020 Ada Interface	
12. PERSONAL AUTHOR(S) Phillip R. Acuff, Wanda M. Hughes, and OAR Corp.	
13a. TYPE OF REPORT Final	13b. TIME COVERED FROM 6/89 TO 2/92
14. DATE OF REPORT (*Year, Month, Day*) 1992, May	15. PAGE COUNT 72
16. SUPPLEMENTARY NOTATION	

17. COSATI CODES		
FIELD	GROUP	SUB-GROUP

18. SUBJECT TERMS (*Continue on reverse if necessary and identify by block number*)
RTEMS, real-time, executive, heterogeneous, homogeneous, multiprocessing, 68020, microprocessor, Ada, runtime
(Continued on page ii)

19. ABSTRACT (*Continue on reverse if necessary and identify by block number*)

This report describes the Level I Ada Interface for the RTEMS real-time executive for the Telesoft Ada Motorola 68020 cross-compiler and is a supplemental document to the Real-Time Executive for Missile Systems User's Guide. This document describes the Ada calling sequence of each RTEMS directive. In addition, an appendix to this document contains the package specification for the RTEMS interface package. The RTEMS Ada language interface provides two classes of packages to the application developer. The first is the package which provides access to RTEMS facilities. The second class of packages provides routines commonly needed by embedded applications.

RTEMS is a real-time executive (kernel) which provides a high performance environment for embedded military applications including such features as multitasking capabilities; homogeneous and heterogeneous multiprocessor systems; event-driven, priority-based, preemptive
(Continued on page ii)

20. DISTRIBUTION / AVAILABILITY OF ABSTRACT ☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT. ☐ DTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION UNCLASSIFIED
22a. NAME OF RESPONSIBLE INDIVIDUAL Wanda M. Hughes	22b. TELEPHONE (*Include Area Code*) (205) 876-4484 22c. OFFICE SYMBOL AMSMI-RD-GC-S

DD Form 1473, JUN 86 *Previous editions are obsolete.*

BLOCK 18 (Cont'd): directive, multitasking, event-driven, priority-based, preemptive, scheduling, intertask communication, synchronization, dynamic memory allocation, user configurable, kernel, embedded, semaphore, events, interrupt, regions, segments, I/O, messages, user extendable, object oriented

BLOCK 19 (Cont'd): scheduling; intertask communication and synchronization; responsive interrupt management; dynamic memory allocation; and a high level of user configurability. RTEMS was originally developed in an effort to eliminate many of the major drawbacks of the Ada programming language. RTEMS provides full capabilities for management of tasks, interrupts, time, and multiprocessors in addition to those features typical of generic operating systems. The code is Government owned, so no licensing fees are necessary. The executive is written using the 'C' programming language with a small amount of assembly language code. The code was developed as a linkable and/or ROMable library with the Ada programming language. Initially RTEMS was developed for the Motorola 68000 family of processors. It has since been ported to the Intel 80386 and 80960 families. This manual describes the implementation of RTEMS for the MC68020 microprocessor for applications using the Ada programming language. Related documents include: Real Time Executive for Missile Systems User's Guide MC68020 'C' Interface, Real Time Executive for Missile Systems MC68020 Timing Document, and Real Time Executive for Missile Systems MC68020 Assembly Interface. RTEMS documentation and code is available for the Motorola 68000 family, and the Intel 80386 and 80960 family of processors.

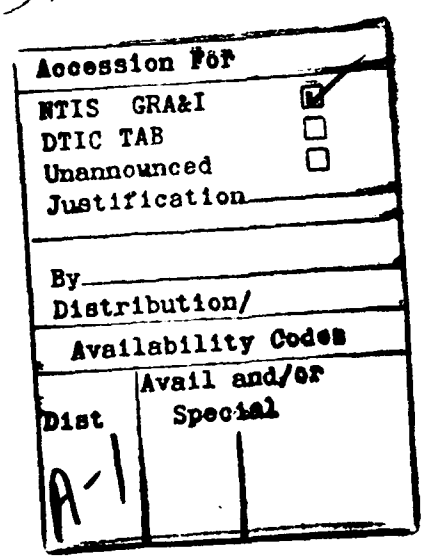


Table of Contents

S.1 INTRODUCTION

S.1.1 Description

This supplemental document describes the Level I Ada Interface for the RTEMS real-time executive for the Telesoft Ada Motorola 68020 cross-compiler. This document describes the Ada calling sequence of each **RTEMS** directive. In addition, an appendix to this document contains the package specification for the **RTEMS** interface package. For more detailed information regarding the exact operation of each directive and its arguments as well as related constants and data structures, please refer to the **RTEMS User's Guide** for a description of that directive and manager.

RTEMS Ada application developers should be aware of the following:

- **AS_ENTER, AS_RETURN, I_ENTER,** *and* **I_RETURN** *are not accessible from an Ada program. They can only be accessed through assembly language routines.*
- *If the application has more than one entry in the Initialization Task Table or the Device Driver Table, then an array of table entries should be declared. If there is only one entry, then a simple constant record declaration will suffice. In either case, the address of the initialized table must be placed in the Configuration Table.*

S.1.2 Packages

The **RTEMS** Ada language interface provides two classes of packages to the application developer. The first is the package which provides access to **RTEMS** facilities. The second class of packages provides routines commonly needed by embedded applications.

S.1.3 RTEMS Package

RTEMS is distributed as a single **Ada** package, **RTEMS**, and a single object module, **RTEMS_CORE**. **RTEMS** resources are made available to Ada compilation units by referencing these resources using a *with* clause and optionally a *use* clause. The **RTEMS** package contains the following items:

- *type definitions*
- *NULL pointer definitions*
- *directive status codes*
- *the date and time record*
- *attribute and option*
- *task mode definitions*
- *event and signal sets*
- *task manager related definitions*
- *task control block (TCB) record*
- *configuration table records*
- Ada *language interface prototypes*

S.1.4 Support Packages

The following list of packages provide functionality commonly needed by embedded Ada applications. The source for these packages is available upon request:

PACKAGE NAME	DESCRIPTION
BITWISE	Boolean Functions
TASK_EXIT	Task Exited Extension

S.2 INITIALIZATION MANAGER

S.2.1 INIT_EXEC - Initialize RTEMS

CALLING SEQUENCE:

```
procedure init_exec (
   conftbl:       in CONFIG_TBL_PTR       -- pointer to configuration table
);
```

NOTES:

This directive does not return to the caller.

S.3 TASK MANAGER

S.3.1 T_CREATE - Create a task

CALLING SEQUENCE:

```
procedure t_create (
    name:       in OBJ_NAME;          -- user-defined four byte name
    priority:   in TASK_PRI;          -- task priority
    stksize:    in UNSIGNED32;        -- stack size (in bytes)
    mode:       in TASK_MODE;         -- task execution mode
    attr:       in UNSIGNED32;        -- task attributes
    tid:        out OBJ_ID;           -- task id
    status:     out DIR_STATUS        -- directive status
);
```

S.3.2 T_IDENT - Get ID of a task

CALLING SEQUENCE:

```
procedure t_ident (
    name:       in OBJ_NAME;          -- user-defined name to search for
    node.       in UNSIGNED32;        -- node(s) to search
    tid:        out OBJ_ID;           -- task id
    status:     out DIR_STATUS        -- directive status
);
```

S.3.3 T_START - Start a task

CALLING SEQUENCE:

```
procedure t_start (
    tid:        in OBJ_ID;                -- task id
    saddr:      in SYSTEM.ADDRESS;        -- task's starting address
    arg:        in UNSIGNED32;            -- initial argument
    status:     out DIR_STATUS            -- directive status
);
```

S.3.4 T_RESTART - Restart a task

CALLING SEQUENCE:

```
procedure t_restart (
    tid:        in OBJ_ID;                -- task id
    arg:        in UNSIGNED32;            -- pointer to argument list
    status:     out DIR_STATUS            -- directive status
);
```

S.3.5 T_DELETE - Delete a task

CALLING SEQUENCE:

```
procedure t_delete (
    tid:        in OBJ_ID;              -- task id
    status:     out DIR_STATUS          -- directive status
);
```

S.3.6 T_SUSPEND - Suspend a task

CALLING SEQUENCE:

```
procedure t_suspend (
    tid:        in OBJ_ID;              -- task id
    status:     out DIR_STATUS          -- directive status
);
```

S.3.7 T_RESUME - Resume a task

CALLING SEQUENCE:

```
procedure t_resume (
    tid:        in OBJ_ID;          -- task id
    status:     out DIR_STATUS      -- directive status
);
```

S.3.8 T_SETPRI - Set a task's priority

CALLING SEQUENCE:

```
procedure t_setpri (
    tid:        in OBJ_ID;          -- task id
    priority:   in TASK_PRI;        -- new task priority
    ppriority:  out TASK_PRI;       -- previous task priority
    status:     out DIR_STATUS      -- directive status
);
```

S.3.9 T_MODE - Change current task's mode

CALLING SEQUENCE:

```
procedure t_mode (
    mode:       in TASK_MODE;          -- new task mode
    mask:       in UNSIGNED32;         -- mode components to alter
    pmode:      out TASK_MODE;         -- previous mode
    status:     out DIR_STATUS         -- directive status
);
```

S.3.10 T_GETNOTE - Get a task's notepad entry

CALLING SEQUENCE:

```
procedure t_getnote (
    tid:        in OBJ_ID;             -- task id
    notepad:    in UNSIGNED32;         -- notepad number
    note:       out UNSIGNED32;        -- notepad value
    status:     out DIR_STATUS         -- directive status
);
```

S.3.11 T_SETNOTE - Set a task's notepad entry

CALLING SEQUENCE:

```
procedure t_setnote (
    tid:        in OBJ_ID;              -- task id
    notepad:    in UNSIGNED32;          -- notepad number
    note:       in UNSIGNED32;          -- notepad value
    status:     out DIR_STATUS          -- directive status
);
```

S.4 INTERRUPT MANAGER

S.4.1 I_ENTER - Enter an ISR

CALLING SEQUENCE:

This directive is accessible only from assembly language.

S.4.2 I_RETURN - Return from an ISR

CALLING SEQUENCE:

This directive is accessible only from assembly language.

S.5 TIME MANAGER

S.5.1 TM_SET - Set system date and time

CALLING SEQUENCE:

```
procedure tm_set (
    timebuf:      in TIME_PTR;          -- pointer to time_info record
    status:       out DIR_STATUS        -- directive status
);
```

S.5.2 TM_GET - Get system date and time

CALLING SEQUENCE:

```
procedure tm_get (
    timebuf:      in TIME_PTR;          -- pointer to time_info record
    status:       out DIR_STATUS        -- directive status
);
```

S.5.3 TM_WKAFTER - Wake up after Interval

CALLING SEQUENCE:

```
procedure tm_wkafter (
    ticks:      in INTERVAL;          -- number of ticks
    status:     out DIR_STATUS        -- directive status
);
```

S.5.4 TM_WKWHEN - Wake up when specified

CALLING SEQUENCE:

```
procedure tm_wkwhen (
    timebuf:    in TIME_PTR;          -- pointer to time_info record
    status:     out DIR_STATUS        -- directive status
);
```

S.5.5 TM_EVAFTER - Send event set after interval

CALLING SEQUENCE:

```
procedure tm_evafter (
    ticks:      in INTERVAL;        -- number of ticks until event
    event:      in EVENT_SET;       -- event set
    tmid:       out OBJ_ID;         -- timer id
    status:     out DIR_STATUS      -- directive status
);
```

S.5.6 TM_EVWHEN - Send event set when specified

CALLING SEQUENCE:

```
procedure tm_evwhen (
    timebuf:    in TIME_PTR;        -- pointer to time_info record
    event:      in EVENT_SET;       -- event set
    tmid:       out OBJ_ID;         -- timer id
    status:     out DIR_STATUS      -- directive status
);
```

S.5.7 TM_EVEVERY - Send periodic event set

CALLING SEQUENCE:

```
procedure tm_evevery (
    ticks:      in INTERVAL;        -- ticks between event sets
    event:      in EVENT_SET;       -- event set
    tmid:       out OBJ_ID;         -- timer id
    status:     out DIR_STATUS      -- directive status
);
```

S.5.8 TM_CANCEL - Cancel timer event

CALLING SEQUENCE:

```
procedure tm_cancel (
    tmid:       in OBJ_ID;          -- timer event id
    status:     out DIR_STATUS      -- directive status
);
```

S.5.9 TM_TICK - Announce a clock tick

CALLING SEQUENCE:

```
procedure tm_tick (
    status:         out DIR_STATUS              -- always returns SUCCESSFUL
);
```

S.6 SEMAPHORE MANAGER

S.6.1 SM_CREATE - Create a semaphore

CALLING SEQUENCE:

```
procedure sm_create (
    name:       in OBJ_NAME;            -- user-defined four byte name
    count:      in UNSIGNED32;          -- initial count
    attr:       in UNSIGNED32;          -- attributes of semaphore
    smid:       out OBJ_ID;             -- semaphore id
    status:     out DIR_STATUS          -- directive status
);
```

S.6.2 SM_IDENT - Get ID of a semaphore

CALLING SEQUENCE:

```
procedure sm_ident (
    name:       in OBJ_NAME;            -- user-defined name to search for
    node:       in UNSIGNED32;          -- node(s) to search
    smid:       out OBJ_ID;             -- semaphore id
    status:     out DIR_STATUS          -- directive status
);
```

S.6.3 SM_DELETE - Delete a semaphore

CALLING SEQUENCE:

```
procedure sm_delete (
    smid:       in OBJ_ID;              -- semaphore id
    status:     out DIR_STATUS          -- directive status
);
```

S.6.4 SM_P - Acquire a semaphore

CALLING SEQUENCE:

```
procedure sm_p (
    smid:       in OBJ_ID;              -- semaphore id
    options:    in UNSIGNED32;          -- semaphore acquisition options
    timeout:    in INTERVAL;            -- maximum interval to wait (in ticks)
    status:     out DIR_STATUS          -- directive status
);
```

S.6.5 SM_V - Release a semaphore

CALLING SEQUENCE:

```
procedure sm_v (
    smid:       in OBJ_ID;              -- semaphore id
    status:     out DIR_STATUS          -- directive status
);
```

S.7 MESSAGE MANAGER

S.7.1 Q_CREATE - Create a queue

CALLING SEQUENCE:

```
procedure q_create (
    name:       in OBJ_NAME;          -- user-defined four byte name
    count:      in UNSIGNED32;        -- maximum message count
    attr:       in UNSIGNED32;        -- queue attributes
    qid:        out OBJ_ID;           -- message queue id
    status:     out DIR_STATUS        -- directive status
);
```

S.7.2 Q_IDENT - Get ID of a queue

CALLING SEQUENCE:

```
procedure q_ident (
    name:       in OBJ_NAME;          -- user-defined name to search for
    node:       in UNSIGNED32;        -- node(s) to search
    qid:        out OBJ_ID;           -- message queue id
    status:     out DIR_STATUS        -- directive status
);
```

S.7.3 Q_DELETE - Delete a queue

CALLING SEQUENCE:

```
procedure q_delete (
    qid:        in OBJ_ID;              -- queue id
    status:     out DIR_STATUS          -- directive status
);
```

S.7.4 Q_SEND - Put message at rear of a queue

CALLING SEQUENCE:

```
procedure q_send (
    qid:        in OBJ_ID;              -- queue id
    buffer:     in SYSTEM.ADDRESS;      -- address of message buffer
    status:     out DIR_STATUS          -- directive status
);
```

S.7.5 Q_URGENT - Put message at front of a queue

CALLING SEQUENCE:

```
procedure q_urgent (
    qid:        in OBJ_ID;              -- queue id
    buffer:     in SYSTEM.ADDRESS;      -- address of message buffer
    status:     out DIR_STATUS          -- directive status
);
```

S.7.6 Q_BROADCAST - Broadcast N messages to a queue

CALLING SEQUENCE:

```
procedure q_broadcast (
    qid:        in OBJ_ID;              -- queue id
    buffer:     in SYSTEM.ADDRESS;      -- address of message buffer
    count:      out UNSIGNED32;         -- number of tasks made ready
    status:     out DIR_STATUS          -- directive status
);
```

S.7.7 Q_RECEIVE - Receive message from a queue

CALLING SEQUENCE:

```
procedure q_receive (
    qid:        in OBJ_ID;                -- queue id
    buffer:     in SYSTEM.ADDRESS;        -- address of message buffer
    options:    in UNSIGNED32;            -- message receive options
    timeout:    in INTERVAL;              -- maximum interval to wait (in ticks)
    status:     out DIR_STATUS            -- directive status
);
```

S.7.8 Q_FLUSH - Flush all messages on a message queue

CALLING SEQUENCE:

```
procedure q_flush (
    qid:        in OBJ_ID;                -- queue id
    count:      out UNSIGNED32            -- number of msgs flushed
    status:     out DIR_STATUS            -- directive status
);
```

S.8 EVENT MANAGER

S.8.1 EV_SEND - Send event set to a task

CALLING SEQUENCE:

```
procedure ev_send (
    tid:        in OBJ_ID;          -- task id to send events to
    event:      in EVENT_SET;       -- event set to send
    status:     out DIR_STATUS      -- directive status
);
```

S.8.2 EV_RECEIVE - Receive event condition

CALLING SEQUENCE:

```
procedure ev_receive (
    eventin:    in EVENT_SET;       -- input condition
    options:    in UNSIGNED32;      -- event receive options
    timeout:    in INTERVAL;        -- maximum interval to wait (in ticks)
    eventout:   out EVENT_SET;      -- output events
    status:     out DIR_STATUS      -- directive status
);
```

S.9 SIGNAL MANAGER

S.9.1 AS_CATCH - Establish an ASR

CALLING SEQUENCE:

```
procedure as_catch (
    asraddr:    in SYSTEM.ADDRESS;    -- address of ASR
    mode:       in TASK_MODE;         -- mode of ASR
    status:     out DIR_STATUS        -- directive status
);
```

S.9.2 AS_SEND - Send signal set to a task

CALLING SEQUENCE:

```
procedure as_send (
    tid:        in OBJ_ID;            -- task id
    signal:     in SIGNAL_SET;        -- signal set
    status:     out DIR_STATUS        -- directive status
);
```

S.9.3 AS_ENTER - Enter an ASR

CALLING SEQUENCE:

This directive is accessible only from assembly language.

S.9.4 AS_RETURN - Return from an ASR

CALLING SEQUENCE:

This directive is accessible only from assembly language.

S.10 PARTITION MANAGER

S.10.1 PT_CREATE - Create a partition

CALLING SEQUENCE:

```
procedure pt_create (
    name:       in OBJ_NAME;            -- user-defined four byte name
    paddr:      in SYSTEM.ADDRESS;      -- physical start address of partition
    length:     in UNSIGNED32;          -- physical length (in bytes)
    bsize:      in UNSIGNED32;          -- size of buffer (in bytes)
    attr:       in UNSIGNED32;          -- partition attributes
    ptid:       out OBJ_ID;             -- partition id
    status:     out DIR_STATUS          -- directive status
);
```

S.10.2 PT_IDENT - Get ID of a partition

CALLING SEQUENCE:

```
procedure pt_ident (
    name:       in OBJ_NAME;            -- user-defined name to search for
    node:       in UNSIGNED32;          -- node(s) to search
    ptid:       out OBJ_ID;             -- partition id
    status:     out DIR_STATUS          -- directive status
);
```

S.10.3 PT_DELETE - Delete a partition

CALLING SEQUENCE:

```
procedure pt_delete (
    ptid:       in OBJ_ID;              -- partition id
    status:     out DIR_STATUS          -- directive status
);
```

S.10.4 PT_GETBUF - Get buffer from a partition

CALLING SEQUENCE:

```
procedure pt_getbuf (
    ptid:       in OBJ_ID;              -- partition id
    bufaddr:    out SYSTEM.ADDRESS;     -- buffer address
    status:     out DIR_STATUS          -- directive status
);
```

S.10.5 PT_RETBUF - Return buffer to a partition

CALLING SEQUENCE:

```
procedure pt_retbuf (
    ptid:       in OBJ_ID;                -- partition id
    bufaddr:    in SYSTEM.ADDRESS;        -- buffer start address
    status:     out DIR_STATUS            -- directive status
);
```

S.11 REGION MANAGER

S.11.1 RN_CREATE - Create a region

CALLING SEQUENCE:

```
procedure rn_create (
    name:       in OBJ_NAME;            -- user-defined four byte name
    paddr:      in SYSTEM.ADDRESS;      -- physical start address of region
    length:     in UNSIGNED32;          -- physical length (in bytes)
    pagesize:   in UNSIGNED32;          -- region page size (in bytes)
    attr:       in UNSIGNED32;          -- region attributes
    rnid:       out OBJ_ID;             -- region id
    status:     out DIR_STATUS          -- directive status
);
```

S.11.2 RN_IDENT - Get ID of a region

CALLING SEQUENCE:

```
procedure rn_ident (
    name:       in OBJ_NAME;            -- user-defined name to search for
    rnid:       out OBJ_ID;             -- region id
    status:     out DIR_STATUS          -- directive status
);
```

S.11.3 RN_DELETE - Delete a region

CALLING SEQUENCE:

```
procedure rn_delete (
    rnid:       in OBJ_ID;              -- region id
    status:     out DIR_STATUS          -- directive status
);
```

S.11.4 RN_GETSEG - Get segment from a region

CALLING SEQUENCE:

```
procedure rn_getseg (
    rnid:       in OBJ_ID;              -- region id
    size:       in UNSIGNED32;          -- segment size desired (in bytes)
    options:    in UNSIGNED32;          -- get segment options
    timeout:    in INTERVAL;            -- maximum interval to wait (in ticks)
    segaddr:    out SYSTEM.ADDRESS;     -- segment start address
    status:     out DIR_STATUS          -- directive status
);
```

S.11.5 RN_RETSEG - Return segment to a region

CALLING SEQUENCE:

```
procedure rn_retseg (
    rnid:       in OBJ_ID;              -- region id
    segaddr:    in SYSTEM.ADDRESS;      -- segment start address
    status:     out DIR_STATUS          -- directive status
);
```

S.12 DUAL-PORT MEMORY MANAGER

S.12.1 DP_CREATE - Create a port

CALLING SEQUENCE:

```
procedure dp_create (
    name:       in OBJ_NAME;              -- user-defined four byte name
    intaddr:    in SYSTEM.ADDRESS;        -- internal start address of region
    extaddr:    in SYSTEM.ADDRESS;        -- external start address of region
    length:     in UNSIGNED32;            -- physical length (in bytes)
    dpid:       out OBJ_ID;               -- dual ported memory port id
    status:     out DIR_STATUS            -- directive status
);
```

S.12.2 DP_IDENT - Get ID of a port

CALLING SEQUENCE:

```
procedure dp_ident (
    name:       in OBJ_NAME;              -- user-defined name to search for
    dpid:       out OBJ_ID;               -- dual ported memory port id
    status:     out DIR_STATUS            -- directive status
);
```

S.12.3 DP_DELETE - Delete a port

CALLING SEQUENCE:

```
procedure dp_delete (
    dpid:       in OBJ_ID;              -- port id
    status:     out DIR_STATUS          -- directive status
);
```

S.12.4 DP_2INTERNAL - Convert external to internal address

CALLING SEQUENCE:

```
procedure dp_2internal (
    dpid:       in OBJ_ID;              -- port id
    external:   in SYSTEM.ADDRESS;      -- external address
    internal:   out SYSTEM.ADDRESS;     -- internal address
    status:     out DIR_STATUS          -- directive status
);
```

S.12.5 DP_2EXTERNAL - Convert internal to external address

CALLING SEQUENCE:

```
procedure dp_2external (
    dpid:       in OBJ_ID;                -- port id
    internal:   in SYSTEM.ADDRESS;        -- internal address
    external:   out SYSTEM.ADDRESS;       -- external address
    status:     out DIR_STATUS            -- directive status
);
```

S.13 INPUT/OUTPUT MANAGER

S.13.1 DE_INIT - Initialize a device driver

CALLING SEQUENCE:

```
procedure de_init (
    dev:        in  UNSIGNED32;          -- 32-bit device number
    argp:       in  SYSTEM.ADDRESS;      -- address of parameter block
    rval:       out UNSIGNED32;          -- return value from the driver
    status:     out DIR_STATUS           -- directive status
);
```

S.13.2 DE_OPEN - Open a device

CALLING SEQUENCE:

```
procedure de_open (
    dev:        in  UNSIGNED32;          -- 32-bit device number
    argp:       in  SYSTEM.ADDRESS;      -- address of parameter block
    rval:       out UNSIGNED32;          -- return value from driver
    status:     out DIR_STATUS           -- directive status
);
```

S.13.3 DE_CLOSE - Close a device

CALLING SEQUENCE:

```
procedure de_close (
    dev:        in  UNSIGNED32;          -- 32-bit device number
    argp:       in  SYSTEM.ADDRESS;      -- address of a parameter block
    rval:       out UNSIGNED32;          -- return value from driver
    status:     out DIR_STATUS           -- directive status
);
```

S.13.4 DE_READ - Read from a device

CALLING SEQUENCE:

```
procedure de_read (
    dev:        in  UNSIGNED32;          -- 32-bit device number
    argp:       in  SYSTEM.ADDRESS;      -- address of a parameter block
    rval:       out UNSIGNED32;          -- return value from driver
    status:     out DIR_STATUS           -- directive status
);
```

S.13.5 DE_WRITE - Write to a device

CALLING SEQUENCE:

```
procedure de_write (
    dev:        in  UNSIGNED32;         -- 32-bit device number
    argp:       in  SYSTEM.ADDRESS;     -- address of a parameter block
    rval:       out UNSIGNED32;         -- return value from driver
    status:     out DIR_STATUS          -- directive status
);
```

S.13.6 DE_CNTRL - Special device services

CALLING SEQUENCE:

```
procedure de_cntrl (
    dev:        in  UNSIGNED32;         -- 32-bit device number
    argp:       in  SYSTEM.ADDRESS;     -- address of a parameter block
    rval:       out UNSIGNED32;         -- return value from driver
    status:     out DIR_STATUS          -- directive status
);
```

S.14 FATAL ERROR MANAGER

S.14.1 K_FATAL - Invoke the fatal error handler

CALLING SEQUENCE:

```
procedure k_fatal (
    errcode:      in UNSIGNED32              -- fatal error code
);
```

NOTES:

This directive does not return to the caller.

S.15 MULTIPROCESSING MANAGER

S.15.1 MP_ANNOUNCE - Announce the arrival of a packet

CALLING SEQUENCE:

procedure mp_announce;

S.16 CREATING AN APPLICATION

S.16.1 Introduction

This appendix demonstrates how to create an RTEMS application using the Telesoft Ada MC680x0 cross-compiler V3.23 as hosted under VMS 5.2. All examples of commands assume that Telesoft Ada is installed and configured correctly.

S.16.2 Environment Requirements

The standard environment modules provided by Telesoft for MC68020 processor boards require minor modifications to support an RTEMS application. The following is a list of requirements for an environment module to support RTEMS:

- *If* RTEMS *time facilities are to be used, a timer must be initialized which will periodically generate an interrupt. The interrupt service routine for the timer should invoke the* tm_tick *directive.*
- *The* RTEMS *Work Space (as specified in the Configuration Table) must be cleared.*
- RTEMS *compatible Input/Output Handlers must be initialized.*
- *Interrupts must be disabled when the* init_exec *directive is invoked.*

S.16.3 Creating Application Library

The following command will create an empty library which can be used to contain the modules which constitute the user's application:

```
$ TSADA/E68/CREATE APPLIB
```

S.16.4 Generating a Default Library File

A default library file is used by the Ada compiler to determine the ordered set of libraries to be searched during reference resolution. The following is an example of a default library file for an RTEMS application:

```
Name: APPLIB
```

```
Name: RTEMS_LIB
Name: RTEMSBSP_LIB
Name: TSADA$DIR:CGS68020
Name: TSADA$DIR:TSA68020RTL
```

The first line indicates the name of the library which will contain the application's compilation units. The second line references the **RTEMS** Ada Interface Library. The third line references the name of the library which contains the environment support routines for the target processor board. The last lines reference the Telesoft Code Generator Support and Run-Time Libraries, respectively.

S.16.5 Compiling an Application

Each compilation unit must be converted into MC68020 object code using a command similar to the following:

$ TSADA/E68/ADA/CPU=MC68020 compilation_unit_name

The above command lists the minimum option set required to generate an MC68020 application. For a list of other supported options, please refer to the **Telesoft TeleGen2 for VAX/VMS Systems to Embedded MC680X0 Targets -- User Guide.**

S.16.6 Binding an Application

The binding process consists of the following activities:

- *verifying that the compilation unit designated as the main program meets all requirements for a main program.*
- *insuring that the main program's context (i.e. imported packages) is current. If any units are missing or out of date, then an error is generated and the binding is aborted.*
- *generating elaboration code and storing it in the application library.*

The main program must be bound with a command similar to the following:

$ TSADA/E68/BIND/CPU=MC68020 compilation_unit_name

The above command lists the minimum option set required to bind an MC68020 Ada application. For a list of other supported options, please refer to the

S.16.7 Generating a Linker Options File

A linker options file is used to specify user environment variables and commonly used linker options. Telesoft provides a number of linker option files which are specifically tailored to support different targets. For more details regarding the linker options file, please refer to the Telesoft TeleGen2 for VAX/VMS Systems to Embedded MC680X0 Targets -- User Guide. The following lines are typically included to provide information required by most RTEMS environment packages:

```
DEFINE/RTEMS_EXEC_RAM=<application dependent>
DEFINE/RTEMS_RAM_SIZE=<application dependent>
DEFINE/RTEMS_MS_TICK=<application dependent>
```

These values must match the corresponding fields in the application configuration table. The environment modules will need these variables to know the location and size of the RTEMS Work Space and the number of milliseconds per clock tick.

In addition, the following lines must be included to allow Telesoft to link with RTEMS:

```
INPUT/OFM RTEMS_BSP
LOCATE/COMPONENT=RTEMS_BSP/OFM/AT=16#003000#
LOCATE/AT=16#4000#
INPUT/OFM RTEMS_CORE
INPUT/OFM RTEMS_IFACE
```

The first line defines the name of the user's board support package module which must be linked with the application. The next line indicates the address at which the user's board support package should be placed by the locator. The third line indicates the location at which other application code may be placed. This address should allow enough room for the entire board support package module. The last two lines are the names of the modules for RTEMS and the Ada interface to RTEMS, respectively.

S.16.8 Linking an Application

The linking process allows users to link compiled Ada programs in preparation for target execution. The linking process resolves references in the following:

- *Ada programs,*
- *bare target run-time support library, and*
- *any imported non-Ada code.*

A command similar to the following is used to link the user's application and create a Motorola S-Record file named **main_prog.sr**.

```
$ TSADA/E68/LINK/SRECORDS/OPTIONS=lk_opt_file main_prog
```

The above command lists the minimum option set required to link an MC68020 Ada application. For a list of other supported options, please refer to the **Telesoft TeleGen2 for VAX/VMS Systems to Embedded MC680X0 Targets -- User Guide.**

S.17 EXAMPLE APPLICATION

```
--  example.ada
--
--  This file contains an example of a simple RTEMS application.
--  It contains a configuration table, a user initialization task,
--  and a simple task.
--
--  This example assumes that a board support package exists
--  and has completed initialization before control is given
--  to the main program.  The example fragment performs some
--  application initialization before initializing RTEMS.
--
--  Most of the tasks in this example are created
--  automatically by RTEMS via the Initialization Task
--  Table.
--

with RTEMS; use RTEMS;

with TEXT_IO;  use TEXT_IO;

procedure EXAMPLE is

Arg: UNSIGNED32:= 0;              -- example tasks ignore args

procedure user_app is

BEGIN

  -- application specific initialization goes here

  LOOP          -- infinite loop
    --  APPLICATION CODE GOES HERE
    --
    --  This code will typically include at least one directive
    --  which causes the calling task to give up the processor.
    --
  END LOOP;

END user_app;
```

```
procedure init_task is
  APP_NAME:   constant OBJ_NAME:= 1;       -- uniqueness helps
  app_tid:    OBJ_ID;
  status:     DIR_STATUS;
begin

  -- example assumes SUCCESSFUL return value

  t_create( APP_NAME, 1, 1024, 0, DEFAULTS, app_tid, status );
  t_start( app_tid, user_app'ADDRESS, Arg, status );
  t_delete( SELF, status );

end init_task;

init_task_tbl: ITASKS_INFO:= (
    STR_TO_OBJ( "ABC " ),        -- init task name "ABC "
    1024,                        -- init task stack size
    1,                           -- init task priority
    DEFAULTS,                    -- init task attributes
    init_task'ADDRESS,           -- init task entry point
    TSLICE,                      -- init task initial mode
    Arg                          -- init task argument list
 );

Config_table: CONFIG_TBL_PTR:= new config_tbl'(
  16#0f0000#,                 -- executive RAM work area
  65536,                      -- executive RAM size
  3,                          -- maximum tasks
  0,                          -- maximum semaphores
  0,                          -- maximum timers
  0,                          -- maximum message queues
  0,                          -- maximum messages
  0,                          -- maximum regions
  0,                          -- maximum partitions
  0,                          -- maximum dp memory areas
  10,                         -- number of ms in a tick
  1,                          -- number of ticks in a timeslice
  1,                          -- number of user init tasks
  init_task_tbl'ADDRESS,      -- user init task(s) table
  0,                          -- number of device drivers
  NULL_DRIVER_TABLE,          -- ptr to driver address table
  NULL_EXT_TABLE,             -- ptr to user extension table
  NULL_MP_TABLE,              -- ptr to MP config table
);

begin
 put_line( "EXAMPLE PROGRAM" );
 init_exec( config_table );
end EXAMPLE;
```

S.18 PACKAGE SPECIFICATION

```
--
-- Package Specification RTEMS ( rtems.ada )
--
-- This package contains information about the
-- executive that is needed by the application.
--

-- ADA Packages

with SYSTEM; use SYSTEM;
with UNCHECKED_CONVERSION;

package RTEMS is

-- processor dependent type definitions

type UNSIGNED8 is new INTEGER range 0..(2**8) - 1;
for UNSIGNED8'size use 8;

type UNSIGNED16 is new LONG_INTEGER range 0..(2**16) - 1;
for UNSIGNED16'size use 16;

type UNSIGNED32 is new LONG_INTEGER range 0..LONG_INTEGER'LAST;

-- RTEMS dependent type definitions
-- RTEMS task, task_ptr, proc_ptr, asr_ptr not supported in ADA

type OBJ_NAME   is new LONG_INTEGER range 0..LONG_INTEGER'LAST;
type OBJ_ID     is new LONG_INTEGER range 0..LONG_INTEGER'LAST;
type DIR_STATUS is new LONG_INTEGER range 0..LONG_INTEGER'LAST;
type TASK_MODE  is new LONG_INTEGER range 0..LONG_INTEGER'LAST;

type TASK_PRI   is new LONG_INTEGER range 0..255;
for TASK_PRI'size use 32;

type INTERVAL   is new LONG_INTEGER range 0..LONG_INTEGER'LAST;
type EVENT_SET  is new LONG_INTEGER range 0..LONG_INTEGER'LAST;
type SIGNAL_SET is new LONG_INTEGER range 0..LONG_INTEGER'LAST;

-- Pointer And address type definitions

function ADDR is new UNCHECKED_CONVERSION( UNSIGNED32, ADDRESS );
```

```
-- RTEMS directive completion statuses

SUCCESSFUL:       constant DIR_STATUS:= 0;  -- successful completion
E_EXITTED:        constant DIR_STATUS:= 1;  -- returned from a task
E_NOMP:           constant DIR_STATUS:= 2;  -- single proc system
E_NAME:           constant DIR_STATUS:= 3;  -- invalid object name
E_ID:             constant DIR_STATUS:= 4;  -- invalid object id
E_TOOMANY:        constant DIR_STATUS:= 5;  -- too many
E_TIMEOUT:        constant DIR_STATUS:= 6;  -- timed out waiting
E_DELETE:         constant DIR_STATUS:= 7;  -- obj deleted while waiting
E_SIZE:           constant DIR_STATUS:= 8;  -- invalid size
E_ADDRESS:        constant DIR_STATUS:= 9;  -- invalid address
E_NUMBER:         constant DIR_STATUS:= 10; -- invalid number
E_NOTDEFINED:     constant DIR_STATUS:= 11; -- item not initialized
E_INUSE:          constant DIR_STATUS:= 12; -- resources outstanding
E_UNSATISFIED:    constant DIR_STATUS:= 13; -- request not satisfied
E_STATE:          constant DIR_STATUS:= 14; -- task is in wrong state
E_ALREADY:        constant DIR_STATUS:= 15; -- task already in state
E_SELF:           constant DIR_STATUS:= 16; -- illegal for calling task
E_REMOTE:         constant DIR_STATUS:= 17; -- illegal on remote object
E_CALLED:         constant DIR_STATUS:= 18; -- incorrect environment
E_PRIORITY:       constant DIR_STATUS:= 19; -- invalid task priority
E_CLOCK:          constant DIR_STATUS:= 20; -- invalid time buffer
E_NODE:           constant DIR_STATUS:= 21; -- invalid node id
E_NOTCONFIGURED:  constant DIR_STATUS:= 22; -- directive not configured
E_NOTIMPLEMENTED:constant DIR_STATUS:= 23; -- directive not implemented

-- Task states

TS_READY:          constant UNSIGNED32:= 16#000#; -- ready to run
TS_DORMANT:        constant UNSIGNED32:= 16#001#; -- created, not started
TS_SUSPEND:        constant UNSIGNED32:= 16#002#; -- wait to be resumed
TS_TRANSIENT:      constant UNSIGNED32:= 16#004#; -- task in transition
TS_DELAY:          constant UNSIGNED32:= 16#008#; -- wait for timeout
TS_WAITSEGMENT:    constant UNSIGNED32:= 16#010#; -- wait for segment
TS_WAITMESSAGE:    constant UNSIGNED32:= 16#020#; -- wait for message
TS_WAITEVENT:      constant UNSIGNED32:= 16#040#; -- wait for event
TS_WAITSEMAPHORE:  constant UNSIGNED32:= 16#080#; -- wait for semaphore
TS_WAITTIME:       constant UNSIGNED32:= 16#100#; -- wait for date/time
TS_WAITRPCREPLY:   constant UNSIGNED32:= 16#200#; -- wait for rpc reply

-- Defaults for attributes or options

DEFAULTS: constant UNSIGNED32:= 16#00000000#; -- all default options


-- Attribute constants

NOFP:      constant UNSIGNED32:= 16#00000000#; -- no floating point unit
```

```
FP:        constant UNSIGNED32:= 16#00000001#; -- floating point unit

LOCAL:     constant UNSIGNED32:= 16#00000000#; -- local object
GLOBAL:    constant UNSIGNED32:= 16#00000002#; -- global object

FIFO:      constant UNSIGNED32:= 16#00000000#; -- process in FIFO order
PRIORITY:  constant UNSIGNED32:= 16#00000004#; -- process by priority

NOLIMIT:   constant UNSIGNED32:= 16#00000000#; -- do not place a limit
LIMIT:     constant UNSIGNED32:= 16#00000008#; -- limit queue entries

-- Options constants

WAIT:      constant UNSIGNED32:= 16#00000000#; -- wait for completion
NOWAIT:    constant UNSIGNED32:= 16#00000001#; -- do not wait on resource

EV_ALL:    constant UNSIGNED32:= 16#00000000#; -- wait for all event(s)
EV_ANY:    constant UNSIGNED32:= 16#00000002#; -- wait for any event(s)

-- Mask constants

PREEMPTMODE: constant UNSIGNED32:= 16#00000100#; -- preemption bit
TSLICEMODE:  constant UNSIGNED32:= 16#00000200#; -- tslice bit
ASRMODE:     constant UNSIGNED32:= 16#00000400#; -- ASR enable bit
INTRMODE:    constant UNSIGNED32:= 16#00000007#; -- execution mode bits

-- Mode constants

PREEMPT:         constant TASK_MODE:= 16#00000000#; -- enable premption
NOPREEMPT:       constant TASK_MODE:= 16#00000100#; -- disable preemption

NOTSLICE:        constant TASK_MODE:= 16#00000000#; -- disable timeslicing
TSLICE:          constant TASK_MODE:= 16#00000200#; -- enable timeslicing

ASR:             constant TASK_MODE:= 16#00000000#; -- enable ASR
NOASR:           constant TASK_MODE:= 16#00000400#; -- disable ASR

function INTR( level: in UNSIGNED32 ) return UNSIGNED32;

-- Null constants (after record definitions in ADA)

-- Identification constants

ALL_NODES:  constant UNSIGNED32:=             0; -- search all nodes
OTHER_NODES:constant UNSIGNED32:= 16#7ffffffe#; -- all except local node
LOCAL_NODE: constant UNSIGNED32:= 16#7fffffff#; -- only local node
WHO_AM_I:   constant UNSIGNED32:=             0; -- calling task

-- Miscellaneous constants
```

```
CURRENT:          constant UNSIGNED32:=     0;   -- current mode/priority
NOTIMEOUT:        constant INTERVAL:=       0;   -- wait indefinitely
SELF:             constant OBJ_ID:=         0;   -- current task
YIELD:            constant INTERVAL:=       0;   -- yield CPU (tm_wkafter)
MIN_PRIORITY:     constant TASK_PRI:=       1;   -- highest task priority
MAX_PRIORITY:     constant TASK_PRI:=     255;   -- lowest task priority
MIN_STK_SIZE:     constant UNSIGNED32:=   256;   -- minimum stack size

-- Definitions for event sets

EVENT_0:     constant EVENT_SET:= 16#1#;
EVENT_1:     constant EVENT_SET:= 16#2#;
EVENT_2:     constant EVENT_SET:= 16#4#;
EVENT_3:     constant EVENT_SET:= 16#8#;
EVENT_4:     constant EVENT_SET:= 16#10#;
EVENT_5:     constant EVENT_SET:= 16#20#;
EVENT_6:     constant EVENT_SET:= 16#40#;
EVENT_7:     constant EVENT_SET:= 16#80#;
EVENT_8:     constant EVENT_SET:= 16#100#;
EVENT_9:     constant EVENT_SET:= 16#200#;
EVENT_10:    constant EVENT_SET:= 16#400#;
EVENT_11:    constant EVENT_SET:= 16#800#;
EVENT_12:    constant EVENT_SET:= 16#1000#;
EVENT_13:    constant EVENT_SET:= 16#2000#;
EVENT_14:    constant EVENT_SET:= 16#4000#;
EVENT_15:    constant EVENT_SET:= 16#8000#;
EVENT_16:    constant EVENT_SET:= 16#10000#;
EVENT_17:    constant EVENT_SET:= 16#20000#;
EVENT_18:    constant EVENT_SET:= 16#40000#;
EVENT_19:    constant EVENT_SET:= 16#80000#;
EVENT_20:    constant EVENT_SET:= 16#100000#;
EVENT_21:    constant EVENT_SET:= 16#200000#;
EVENT_22:    constant EVENT_SET:= 16#400000#;
EVENT_23:    constant EVENT_SET:= 16#800000#;
EVENT_24:    constant EVENT_SET:= 16#1000000#;
EVENT_25:    constant EVENT_SET:= 16#2000000#;
EVENT_26:    constant EVENT_SET:= 16#4000000#;
EVENT_27:    constant EVENT_SET:= 16#8000000#;
EVENT_28:    constant EVENT_SET:= 16#10000000#;
EVENT_29:    constant EVENT_SET:= 16#20000000#;
EVENT_30:    constant EVENT_SET:= 16#40000000#;
-- EVENT_31 is not accessible by ADA programs



-- Definitions for signal sets

SIGNAL_0 : constant SIGNAL_SET:= 16#1#;
```

```
SIGNAL_1 : constant SIGNAL_SET:= 16#2#;
SIGNAL_2 : constant SIGNAL_SET:= 16#4#;
SIGNAL_3 : constant SIGNAL_SET:= 16#8#;
SIGNAL_4 : constant SIGNAL_SET:= 16#10#;
SIGNAL_5 : constant SIGNAL_SET:= 16#20#;
SIGNAL_6 : constant SIGNAL_SET:= 16#40#;
SIGNAL_7 : constant SIGNAL_SET:= 16#80#;
SIGNAL_8 : constant SIGNAL_SET:= 16#100#;
SIGNAL_9 : constant SIGNAL_SET:= 16#200#;
SIGNAL_10: constant SIGNAL_SET:= 16#400#;
SIGNAL_11: constant SIGNAL_SET:= 16#800#;
SIGNAL_12: constant SIGNAL_SET:= 16#1000#;
SIGNAL_13: constant SIGNAL_SET:= 16#2000#;
SIGNAL_14: constant SIGNAL_SET:= 16#4000#;
SIGNAL_15: constant SIGNAL_SET:= 16#8000#;
SIGNAL_16: constant SIGNAL_SET:= 16#10000#;
SIGNAL_17: constant SIGNAL_SET:= 16#20000#;
SIGNAL_18: constant SIGNAL_SET:= 16#40000#;
SIGNAL_19: constant SIGNAL_SET:= 16#80000#;
SIGNAL_20: constant SIGNAL_SET:= 16#100000#;
SIGNAL_21: constant SIGNAL_SET:= 16#200000#;
SIGNAL_22: constant SIGNAL_SET:= 16#400000#;
SIGNAL_23: constant SIGNAL_SET:= 16#800000#;
SIGNAL_24: constant SIGNAL_SET:= 16#1000000#;
SIGNAL_25: constant SIGNAL_SET:= 16#2000000#;
SIGNAL_26: constant SIGNAL_SET:= 16#4000000#;
SIGNAL_27: constant SIGNAL_SET:= 16#8000000#;
SIGNAL_28: constant SIGNAL_SET:= 16#10000000#;
SIGNAL_29: constant SIGNAL_SET:= 16#20000000#;
SIGNAL_30: constant SIGNAL_SET:= 16#40000000#;
-- SIGNAL_31 is not accessible by ADA programs

-- Notepad location constants

NOTEPAD_0:  constant UNSIGNED32:= 0;          -- Notepad Location 0
NOTEPAD_1:  constant UNSIGNED32:=  1;         -- Notepad Location 1
NOTEPAD_2:  constant UNSIGNED32:=  2;         -- Notepad Location 2
NOTEPAD_3:  constant UNSIGNED32:=  3;         -- Notepad Location 3
NOTEPAD_4:  constant UNSIGNED32:=  4;         -- Notepad Location 4
NOTEPAD_5:  constant UNSIGNED32:=  5;         -- Notepad Location 5
NOTEPAD_6:  constant UNSIGNED32:=  6;         -- Notepad Location 6
NOTEPAD_7:  constant UNSIGNED32:=  7;         -- Notepad Location 7
NOTEPAD_8:  constant UNSIGNED32:=  8;         -- Notepad Location 8
NOTEPAD_9:  constant UNSIGNED32:=  9;         -- Notepad Location 9
NOTEPAD_10: constant UNSIGNED32:= 10;         -- Notepad Location 10
NOTEPAD_11: constant UNSIGNED32:= 11;         -- Notepad Location 11
NOTEPAD_12: constant UNSIGNED32:= 12;         -- Notepad Location 12
NOTEPAD_13: constant UNSIGNED32:= 13;         -- Notepad Location 13
NOTEPAD_14: constant UNSIGNED32:= 14;         -- Notepad Location 14
NOTEPAD_15: constant UNSIGNED32:= 15;         -- Notepad Location 15
```

```
-- Multiprocessing constants

MIN_PKTSIZE: constant UNSIGNED32:= 64;  -- MPCI layer must support
                                        -- packets >= this size

-- Macros to access sub-fields in an object id

function Node( obj: in OBJ_ID )   return UNSIGNED32;
pragma inline( Node );

function Object( obj: in OBJ_ID ) return UNSIGNED32;
pragma inline( Object );

subtype OBJ_STR is string( 1..4 );

function OBJ_TO_STR is new unchecked_conversion( OBJ_NAME, OBJ_STR );

function STR_TO_OBJ is new unchecked_conversion( OBJ_STR, OBJ_NAME );

-- Time type Definitions

type YEARS is new INTEGER range 1988..INTEGER'last;

type MONTHS is new INTEGER range 1..12;
for MONTHS'size use 8;

type DAYS is new INTEGER range 1..31;
for DAYS'size use 8;

type HOURS is new INTEGER range 0..23;
for HOURS'size use 16;

type MINUTES is new INTEGER range 0..59;
for MINUTES'size use 8;

type SECONDS is new INTEGER range 0..59;
for SECONDS'size use 8;



-- Date and time record

type TIME_INFO is record
```

```
  year:   YEARS;              -- year, A.D.
  month:  MONTHS;             -- month, 1 -> 12
  day:    DAYS;               -- day, 1 -> 31
  hour:   HOURS;              -- hour, 1 -> 23
  minute: MINUTES;            -- minute, 0 -> 59
  second: SECONDS;            -- second, 0 -> 59
  ticks:  UNSIGNED32;         -- elapsed ticks between secs
end record;

type TIME_PTR is access TIME_INFO;

-- MC68020 registers

type REGISTERS is record
  d0:   UNSIGNED32;                -- data register 0
  d1:   UNSIGNED32;                -- data register 1
  d2:   UNSIGNED32;                -- data register 2
  d3:   UNSIGNED32;                -- data register 3
  d4:   UNSIGNED32;                -- data register 4
  d5:   UNSIGNED32;                -- data register 5
  d6:   UNSIGNED32;                -- data register 6
  d7:   UNSIGNED32;                -- data register 7
  a0:   UNSIGNED32;                -- address register 0
  a1:   UNSIGNED32;                -- address register 1
  a2:   UNSIGNED32;                -- address register 2
  a3:   UNSIGNED32;                -- address register 3
  a4:   UNSIGNED32;                -- address register 4
  a5:   UNSIGNED32;                -- address register 5
  a6:   UNSIGNED32;                -- address register 6
  msp:  UNSIGNED32;                -- master stack pointer
end record;

-- Task control block

type T_CTLBLK;

type T_CB is access T_CTLBLK;

type U8_ARRAY is array( POSITIVE range  ) of UNSIGNED8;

type NP_ARRAY is array( NOTEPAD_0..NOTEPAD_15 ) of UNSIGNED32;



-- Task control block

type T_CTLBLK is record
  next:       T_CB;                -- pointer to next TCB
  prev:       T_CB;                -- pointer to previous TCB
  tid:        OBJ_ID;              -- task identification
```

```
  name:        OBJ_NAME;              -- task name
  state:       UNSIGNED32;            -- task state
  priority:    TASK_PRI;              -- current task priority
  wait:        OBJ_ID;                -- id of resource waiting on
  resvd1:      U8_ARRAY( 1..144 );    -- reserved for RTEMS
  mode:        TASK_MODE;             -- current task mode
  attributes:  UNSIGNED32;            -- task attributes
  regs:        REGISTERS;             -- MC68020 register save area
  fp_context:  ADDRESS;               -- pointer to FP context area
  notepad:     NP_ARRAY;              -- executive notepads
  extension:   ADDRESS;               -- pointer TCB extension
end record;

-- Initialization task(s) record

type ITASKS_INFO is record
  name:        OBJ_NAME;              -- task name
  stksize:     UNSIGNED32;            -- task stack size
  priority:    TASK_PRI;              -- task priority
  attributes:  UNSIGNED32;            -- task flags
  entry_point: ADDRESS;               -- task entry point
  mode:        TASK_MODE;             -- task initial mode
  arg:         UNSIGNED32;            -- task argument
end RECORD;

-- Driver description record

type DRIVER_INFO is record
  init:        ADDRESS;               -- initialization procedure
  open:        ADDRESS;               -- open request procedure
  close:       ADDRESS;               -- close request procedure
  read:        ADDRESS;               -- read request procedure
  write:       ADDRESS;               -- write request procedure
  cntrl:       ADDRESS;               -- control request procedure
  reserved1:   UNSIGNED32;            -- reserved for RTEMS
  reserved2:   UNSIGNED32;            -- reserved for RTEMS
end record;

-- Optional task related extensions

type EXT_INFO is record
  tcreate:     ADDRESS;               -- tcreate user extension
  tstart:      ADDRESS;               -- tstart user extension
  trestart:    ADDRESS;               -- trestart user extension
```

```
  tdelete:      ADDRESS;            -- tdelete user extension
  tswitch:      ADDRESS;            -- tswitch user extension
  taskexitted:  ADDRESS;            -- task exitted user extension
  fatal:        ADDRESS;            -- fatal user extension
end record;

type EXT_TBL_PTR    is access EXT_INFO;

--  Multiprocessor communications address record

type MPCI_INFO is record
  init:          ADDRESS;           --  initialization procedure
  getpkt:        ADDRESS;           --  get packet procedure
  retpkt:        ADDRESS;           --  return packet procedure
  send:          ADDRESS;           --  packet send procedure
  receive:       ADDRESS;           --  packet receive procedure
end record;

type MPCI_TBL_PTR   is access MPCI_INFO;

-- Multiprocessor configuration table

type MP_INFO is record
  node:           UNSIGNED16;       -- local node number
  max_nodes:      UNSIGNED16;       -- number nodes in system
  max_gobjects:   UNSIGNED32;       -- maximum global objects
  max_proxies:    UNSIGNED32;       -- maximum proxies
  Mpci_tbl:       MPCI_TBL_PTR;     -- MPCI table
end record;

type MP_TBL_PTR     is access MP_INFO;

-- Configuration table

type CONFIG_INFO is record
  exec_ram:       UNSIGNED32;        -- RTEMS Workspace
  ram_size:       UNSIGNED32;        -- RTEMS Workspace size
  max_tasks:      UNSIGNED16;        -- max number tasks
  max_semaphores: UNSIGNED16;        -- max number semaphores
  max_timers:     UNSIGNED16;        -- max number timers
  max_queues:     UNSIGNED16;        -- max number queues
  max_messages:   UNSIGNED16;        -- max number messages
  max_regions:    UNSIGNED16;        -- max number regions
  max_partitions: UNSIGNED16;        -- max number partitions
  max_dpmems:     UNSIGNED16;        -- max number dp memory areas
  ms_tick:        UNSIGNED16;        -- ms in a tick
  tslice:         UNSIGNED16;        -- ticks in a time-slice
  num_itasks:     UNSIGNED32;        -- number of user init tasks
  Itasks_tbl:     ADDRESS;           -- init task table
  num_devices:    UNSIGNED32;        -- number of device drivers
```

```
  Drv_tbl:        ADDRESS;            -- device driver table
  Ext_tbl:        EXT_TBL_PTR;        -- extension table
  Mp_tbl:         MP_TBL_PTR;         -- MP config table
end record;

type CONFIG_TBL_PTR is access CONFIG_INFO;

-- Unchecked_conversions to generate RTEMS NULL pointers

function U32_TO_EXTTBL is new
       UNCHECKED_CONVERSION( UNSIGNED32, EXT_TBL_PTR );
function U32_TO_MPCI is new
       UNCHECKED_CONVERSION( UNSIGNED32, MPCI_TBL_PTR );
function U32_TO_MP is new
       UNCHECKED_CONVERSION( UNSIGNED32, MP_TBL_PTR );

-- Null constants (after record definitions in ADA)

NULL_PACKET:        constant ADDRESS:=        ADDR( 0 );
NULL_DP_ADDRESS:    constant ADDRESS:=        ADDR( 0 );
NULL_TASK:          constant ADDRESS:=        ADDR( 0 );
NULL_ASR:           constant ADDRESS:=        ADDR( 0 );
NULL_DRIVER:        constant ADDRESS:=        ADDR( 0 );
NULL_EXTENSION:     constant ADDRESS:=        ADDR( 0 );
NULL_DRIVER_TABLE:  constant ADDRESS:=        ADDR( 0 );
NULL_ITASKS_TABLE:  constant ADDRESS:=        ADDR( 0 );
NULL_EXT_TABLE:     constant EXT_TBL_PTR:=    U32_TO_EXTTBL( 0 );
NULL_MPCI_TABLE:    constant MPCI_TBL_PTR:=   U32_TO_MPCI( 0 );
NULL_MP_TABLE:      constant MP_TBL_PTR:=     U32_TO_MP( 0 );

-- Signal Manager Directives

procedure  as_catch( asraddr: in ADDRESS; mode: in TASK_MODE;
                     status: out DIR_STATUS );
pragma Interface ( assembly, as_catch );
pragma Linkname ( as_catch, "RTEMS_as_catch" );


procedure  as_send( tid: in OBJ_ID; signal: in SIGNAL_SET;
                    status: out DIR_STATUS );
pragma Interface ( assembly, as_send );
pragma Linkname ( as_send, "RTEMS_as_send" );


-- IO Manager Directives

procedure  de_close( dev: in UNSIGNED32; argp: in ADDRESS;
                     rval: out UNSIGNED32; status: out DIR_STATUS );
pragma Interface ( assembly, de_close );
pragma Linkname ( de_close, "RTEMS_de_close" );
```

```
procedure  de_cntrl( dev: in UNSIGNED32; argp: in ADDRESS;
                     rval: out UNSIGNED32; status: out DIR_STATUS );
pragma Interface ( assembly, de_cntrl );
pragma Linkname ( de_cntrl, "RTEMS_de_cntrl" );

procedure  de_init( dev: in UNSIGNED32; argp: in ADDRESS;
                    rval: out UNSIGNED32; status: out DIR_STATUS );
pragma Interface ( assembly, de_init );
pragma Linkname ( de_init, "RTEMS_de_init" );

procedure  de_open( dev: in UNSIGNED32; argp: in ADDRESS;
                    rval: out UNSIGNED32; status: out DIR_STATUS );
pragma Interface ( assembly, de_open );
pragma Linkname ( de_open, "RTEMS_de_open" );

procedure  de_read( dev: in UNSIGNED32; argp: in ADDRESS;
                    rval: out UNSIGNED32; status: out DIR_STATUS );
pragma Interface ( assembly, de_read );
pragma Linkname ( de_read, "RTEMS_de_read" );

procedure  de_write( dev: in UNSIGNED32; argp: in ADDRESS;
                     rval: out UNSIGNED32; status: out DIR_STATUS );
pragma Interface ( assembly, de_write );
pragma Linkname ( de_write, "RTEMS_de_write" );

-- Dual-Ported Memory Manager Directives

procedure  dp_create( name: in OBJ_NAME; intaddr, extaddr: in ADDRESS;
                      length: in UNSIGNED32; dpid: out OBJ_ID;
                      status: out DIR_STATUS );
pragma Interface ( assembly, dp_create );
pragma Linkname ( dp_create, "RTEMS_dp_create" );

procedure  dp_ident( name: in OBJ_NAME; dpid: out OBJ_ID;
                     status: out DIR_STATUS );
pragma Interface ( assembly, dp_ident );
pragma Linkname ( dp_ident, "RTEMS_dp_ident" );

procedure  dp_delete( dpid: in OBJ_ID; status: out DIR_STATUS );
pragma Interface ( assembly, dp_delete );
pragma Linkname ( dp_delete, "RTEMS_dp_delete" );


procedure  dp_2internal( dpid: OBJ_ID; external: in ADDRESS;
                         internal: out ADDRESS; status: out DIR_STATUS);
pragma Interface ( assembly, dp_2internal );
pragma Linkname ( dp_2internal, "RTEMS_dp_2internal" );

procedure  dp_2external( dpid: OBJ_ID; internal: in ADDRESS;
```

```
                        external: out ADDRESS; status: out DIR_STATUS);
pragma Interface ( assembly, dp_2external );
pragma Linkname ( dp_2external, "RTEMS_dp_2external" );

-- Event Manager Directives

procedure  ev_receive( eventin: EVENT_SET; options: in UNSIGNED32;
                       timeout: in INTERVAL; eventout: out EVENT_SET;
                       status: out DIR_STATUS );
pragma Interface ( assembly, ev_receive );
pragma Linkname ( ev_receive, "RTEMS_ev_receive" );

procedure  ev_send( tid: in OBJ_ID; event: in EVENT_SET;
                    status: out DIR_STATUS );
pragma Interface ( assembly, ev_send );
pragma Linkname ( ev_send, "RTEMS_ev_send" );

-- Initialization Manager Directives

procedure  init_exec( conftbl: in CONFIG_TBL_PTR );
pragma Interface ( assembly, init_exec );
pragma Linkname ( init_exec, "RTEMS_init_exec" );

-- Partition Manager Directives

procedure  pt_create( name: in OBJ_NAME; paddr: in ADDRESS;
                      length, bsize: in UNSIGNED32; attr: in UNSIGNED32;
                      ptid: out OBJ_ID; status: out DIR_STATUS );
pragma Interface ( assembly, pt_create );
pragma Linkname ( pt_create, "RTEMS_pt_create" );

procedure  pt_delete( ptid: in OBJ_ID; status: out DIR_STATUS );
pragma Interface ( assembly, pt_delete );
pragma Linkname ( pt_delete, "RTEMS_pt_delete" );

procedure  pt_getbuf( ptid: in OBJ_ID; bufaddr: out ADDRESS;
                      status: out DIR_STATUS );
pragma Interface ( assembly, pt_getbuf );
pragma Linkname ( pt_getbuf, "RTEMS_pt_getbuf" );

procedure  pt_ident( name: in OBJ_NAME; node: in UNSIGNED32;
                     ptid: out OBJ_ID; status: out DIR_STATUS );
pragma Interface ( assembly, pt_ident );
pragma Linkname ( pt_ident, "RTEMS_pt_ident" );

procedure pt_retbuf( ptid: in OBJ_ID; bufaddr: in ADDRESS;
                     status: out DIR_STATUS );
pragma Interface ( assembly, pt_retbuf );
pragma Linkname ( pt_retbuf, "RTEMS_pt_retbuf" );
```

```
-- Message Manager Directives

procedure  q_broadcast( qid: in OBJ_ID; buffer: in ADDRESS;
                        count: out UNSIGNED32; status: out DIR_STATUS );
pragma Interface ( assembly, q_broadcast );
pragma Linkname ( q_broadcast, "RTEMS_q_broadcast" );

procedure  q_create( name: in OBJ_NAME; count: in UNSIGNED32;
                     attr: in UNSIGNED32; qid: out OBJ_ID;
                     status: out DIR_STATUS );
pragma Interface ( assembly, q_create );
pragma Linkname ( q_create, "RTEMS_q_create" );

procedure  q_delete( qid: in OBJ_ID; status: out DIR_STATUS );
pragma Interface ( assembly, q_delete );
pragma Linkname ( q_delete, "RTEMS_q_delete" );

procedure  q_flush( qid: in OBJ_ID; status: out DIR_STATUS );
pragma Interface ( assembly, q_flush );
pragma Linkname ( q_flush, "RTEMS_q_flush" );

procedure  q_ident( name: in OBJ_NAME; node: in UNSIGNED32;
                    qid: out OBJ_ID; status: out DIR_STATUS );
pragma Interface ( assembly, q_ident );
pragma Linkname ( q_ident, "RTEMS_q_ident" );

procedure  q_receive( qid: in OBJ_ID; buffer: in ADDRESS;
                      options: in UNSIGNED32; timeout: in INTERVAL;
                      status: out DIR_STATUS );
pragma Interface ( assembly, q_receive );
pragma Linkname ( q_receive, "RTEMS_q_receive" );

procedure  q_send( qid: in OBJ_ID; buffer: in ADDRESS;
                   status: out DIR_STATUS );
pragma Interface ( assembly, q_send );
pragma Linkname ( q_send, "RTEMS_q_send" );

procedure  q_urgent( qid: in OBJ_ID; buffer: in ADDRESS;
                     status: out DIR_STATUS );
pragma Interface ( assembly, q_urgent );
pragma Linkname ( q_urgent, "RTEMS_q_urgent" );


-- Region Manager Directives

procedure  rn_create( name: in OBJ_NAME; paddr: in ADDRESS;
                      length, pagesize: in UNSIGNED32;
                      attr: in UNSIGNED32; rnid: out OBJ_ID;
                      status: out DIR_STATUS );
```

```
pragma Interface ( assembly, rn_create );
pragma Linkname ( rn_create, "RTEMS_rn_create" );

procedure  rn_delete( rnid: in OBJ_ID; status: out DIR_STATUS );
pragma Interface ( assembly, rn_delete );
pragma Linkname ( rn_delete, "RTEMS_rn_delete" );

procedure  rn_getseg( rnid: in OBJ_ID; size, options: in UNSIGNED32;
                      timeout: in INTERVAL; segaddr: out ADDRESS;
                      status: out DIR_STATUS );
pragma Interface ( assembly, rn_getseg );
pragma Linkname ( rn_getseg, "RTEMS_rn_getseg" );

procedure  rn_ident( name: in OBJ_NAME; rnid: out OBJ_ID;
                     status: out DIR_STATUS );
pragma Interface ( assembly, rn_ident );
pragma Linkname ( rn_ident, "RTEMS_rn_ident" );

procedure  rn_retseg( rnid: in OBJ_ID; segaddr: in ADDRESS;
                      status: out DIR_STATUS );
pragma Interface ( assembly, rn_retseg );
pragma Linkname ( rn_retseg, "RTEMS_rn_retseg" );

-- Semaphore Manager Directives

procedure  sm_create( name: OBJ_NAME; count: in UNSIGNED32;
                      attr: in UNSIGNED32; smid: out OBJ_ID;
                      status: out DIR_STATUS );
pragma Interface ( assembly, sm_create );
pragma Linkname ( sm_create, "RTEMS_sm_create" );

procedure  sm_delete( smid: in OBJ_ID; status: out DIR_STATUS );
pragma Interface ( assembly, sm_delete );
pragma Linkname ( sm_delete, "RTEMS_sm_delete" );

procedure  sm_ident( name: in OBJ_NAME; node: in UNSIGNED32;
                     smid: out OBJ_ID; status: out DIR_STATUS );
pragma Interface ( assembly, sm_ident );
pragma Linkname ( sm_ident, "RTEMS_sm_ident" );

procedure  sm_p( smid: in OBJ_ID; options: in UNSIGNED32;
                 timeout: in INTERVAL; status: out DIR_STATUS );
pragma Interface ( assembly, sm_p);
pragma Linkname ( sm_p, "RTEMS_sm_p" );

procedure  sm_v( smid: in OBJ_ID; status: out DIR_STATUS );
pragma Interface ( assembly, sm_v );
pragma Linkname ( sm_v, "RTEMS_sm_v" );

-- Task Manager Directives
```

```
procedure  t_create( name: in OBJ_NAME; priority: in TASK_PRI;
                     stksize: in UNSIGNED32; mode: in TASK_MODE;
                     attr: in UNSIGNED32; tid: out OBJ_ID;
                     status: out DIR_STATUS );
pragma Interface ( assembly, t_create );
pragma Linkname ( t_create, "RTEMS_t_create" );

procedure  t_delete( tid: in OBJ_ID; status: out DIR_STATUS );
pragma Interface ( assembly, t_delete );
pragma Linkname ( t_delete, "RTEMS_t_delete" );

procedure  t_getnote( tid: in OBJ_ID; notepad: in UNSIGNED32;
                      note: out UNSIGNED32; status: out DIR_STATUS );
pragma Interface ( assembly, t_getnote );
pragma Linkname ( t_getnote, "RTEMS_t_getnote" );

procedure  t_ident( name: in OBJ_NAME; node: in UNSIGNED32;
                    tid: out OBJ_ID; status: out DIR_STATUS );
pragma Interface ( assembly, t_ident );
pragma Linkname ( t_ident, "RTEMS_t_ident" );

procedure  t_mode( mode: in TASK_MODE; mask: in UNSIGNED32;
                   pmode: out TASK_MODE; status: out DIR_STATUS );
pragma Interface ( assembly, t_mode );
pragma Linkname ( t_mode, "RTEMS_t_mode" );

procedure  t_restart( tid: in OBJ_ID; arg: in UNSIGNED32;
                      status: out DIR_STATUS );
pragma Interface ( assembly, t_restart );
pragma Linkname ( t_restart, "RTEMS_t_restart" );

procedure  t_resume( tid: in OBJ_ID; status: out DIR_STATUS );
pragma Interface ( assembly, t_resume );
pragma Linkname ( t_resume, "RTEMS_t_resume" );

procedure  t_setpri( tid: in OBJ_ID; priority: in TASK_PRI;
                     ppriority: out TASK_PRI; status: out DIR_STATUS );
pragma Interface ( assembly, t_setpri );
pragma Linkname ( t_setpri, "RTEMS_t_setpri" );



procedure  t_setnote( tid: in OBJ_ID; notepad, note: in UNSIGNED32;
                      status: out DIR_STATUS );
pragma Interface ( assembly, t_setnote );
pragma Linkname ( t_setnote, "RTEMS_t_setnote" );

procedure  t_start( tid:in OBJ_ID; saddr: in ADDRESS; arg:in UNSIGNED32;
```

```
                    status: out DIR_STATUS );
pragma Interface ( assembly, t_start );
pragma Linkname ( t_start, "RTEMS_t_start" );

procedure  t_suspend( tid: in OBJ_ID; status: out DIR_STATUS );
pragma Interface ( assembly, t_suspend );
pragma Linkname ( t_suspend, "RTEMS_t_suspend" );

-- Time Manager Directives

procedure  tm_cancel( tmid: in OBJ_ID; status: out DIR_STATUS );
pragma Interface ( assembly, tm_cancel );
pragma Linkname ( tm_cancel, "RTEMS_tm_cancel" );

procedure  tm_evafter( ticks: in INTERVAL; event: in EVENT_SET;
                       tmid: out OBJ_ID; status: out DIR_STATUS );
pragma Interface ( assembly, tm_evafter );
pragma Linkname ( tm_evafter, "RTEMS_tm_evafter" );

procedure  tm_evevery( ticks: in INTERVAL; event: in EVENT_SET;
                       tmid: out OBJ_ID; status: out DIR_STATUS );
pragma Interface ( assembly, tm_evevery );
pragma Linkname ( tm_evevery, "RTEMS_tm_evevery" );

procedure  tm_evwhen( timebuf: in TIME_PTR; event: in EVENT_SET;
                      tmid: out OBJ_ID;  status: out DIR_STATUS );
pragma Interface ( assembly, tm_evwhen );
pragma Linkname ( tm_evwhen, "RTEMS_tm_evwhen" );

procedure  tm_get( timebuf: in TIME_PTR; status: out DIR_STATUS );
pragma Interface ( assembly, tm_get );
pragma Linkname ( tm_get, "RTEMS_tm_get" );

procedure  tm_set( timebuf: in TIME_PTR; status: out DIR_STATUS );
pragma Interface ( assembly, tm_set );
pragma Linkname ( tm_set, "RTEMS_tm_set" );

procedure  tm_tick( status: out DIR_STATUS );
pragma Interface ( assembly, tm_tick );
pragma Linkname ( tm_tick, "RTEMS_tm_tick" );

procedure  tm_wkafter( ticks: in INTERVAL; status: out DIR_STATUS );
pragma Interface ( assembly, tm_wkafter );
pragma Linkname ( tm_wkafter, "RTEMS_tm_wkafter" );

procedure  tm_wkwhen( timebuf: in TIME_PTR; status: out DIR_STATUS );
pragma Interface ( assembly, tm_wkwhen );
pragma Linkname ( tm_wkwhen, "RTEMS_tm_wkwhen" );

-- Error Manager Directives
```

```
procedure  k_fatal( errcode: in UNSIGNED32 );
pragma Interface ( assembly, k_fatal );
pragma Linkname ( k_fatal, "RTEMS_k_fatal" );

-- Multi-Processing Directives

procedure  mp_announce;
pragma Interface ( assembly, mp_announce );
pragma Linkname ( mp_announce, "RTEMS_mp_announce" );

end RTEMS;
```

INITIAL DISTRIBUTION

	Copies
U.S. Army Materiel System Analysis Activity ATTN: AMXSY-MP (Herbert Cohen) Aberdeen Proving Ground, MD 21005	1
IIT Research Institute ATTN: GACIAC 10 W. 35th Street Chicago, IL 60616	1
WL/MNAG ATTN: Chris Anderson Eglin AFB, FL 32542-5434	1
Naval Weapons Center Missile Software Technology Office Code 3901C, ATTN: Mr. Carl W. Hall China Lake, CA 93555-6001	1
On-line Applications Research 3315 Memorial Parkway, SW Huntsville, AL 35801	3
CEA Incorporated Blue Hills Office Park 150 Royall Street Suite 260, ATTN: Mr. John Shockro Canton, MA 01021	1
VITA 10229 N. Scottsdale Rd Suite B, ATTN: Mr. Ray Alderman Scottsdale, AZ 85253	1
Westinghouse Electric Corp. P.O. Box 746 - MS432 ATTN: Mr. Eli Solomon Baltimore, MD 21203	1
Dept. of Computer Science B-173 Florida State University ATTN: Dr. Ted Baker Tallahassee, FL 32306-4019	1
DSD Laboratories 75 Union Avenue ATTN: Mr. Roger Whitehead Studbury, MA 01776	1

	Copies
AMSMI-RD	1
AMSMI-RD-GS, Dr. Paul Jacobs	1
AMSMI-RD-GC-S, Gerald E. Scheiman	1
Wanda M. Hughes	5
Phillip Acuff	4
AMSMI-RD-BA	1
AMSMI-RD-BA-C3, Bob Christian	1
AMSMI-RD-SS	1
AMSMI-RD-CS-R	15
AMSMI-RD-CS-T	1
AMSMI-GC-IP, Mr. Fred M. Bush	1
CSSD-CR-S, Mr. Frank Poslajko	1
SFAE-FS-ML-TM, Mr. Frank Gregory	1
SFAE-AD-ATA-SE, Mr. Julian Cothran	1
Mr. John Carter	1

☆ U.S. GOVERNMENT PRINTING OFFICE 1992 631-100/60225